Prédiction du remboursement des prêts

Réalisé par : Nazim KESKES

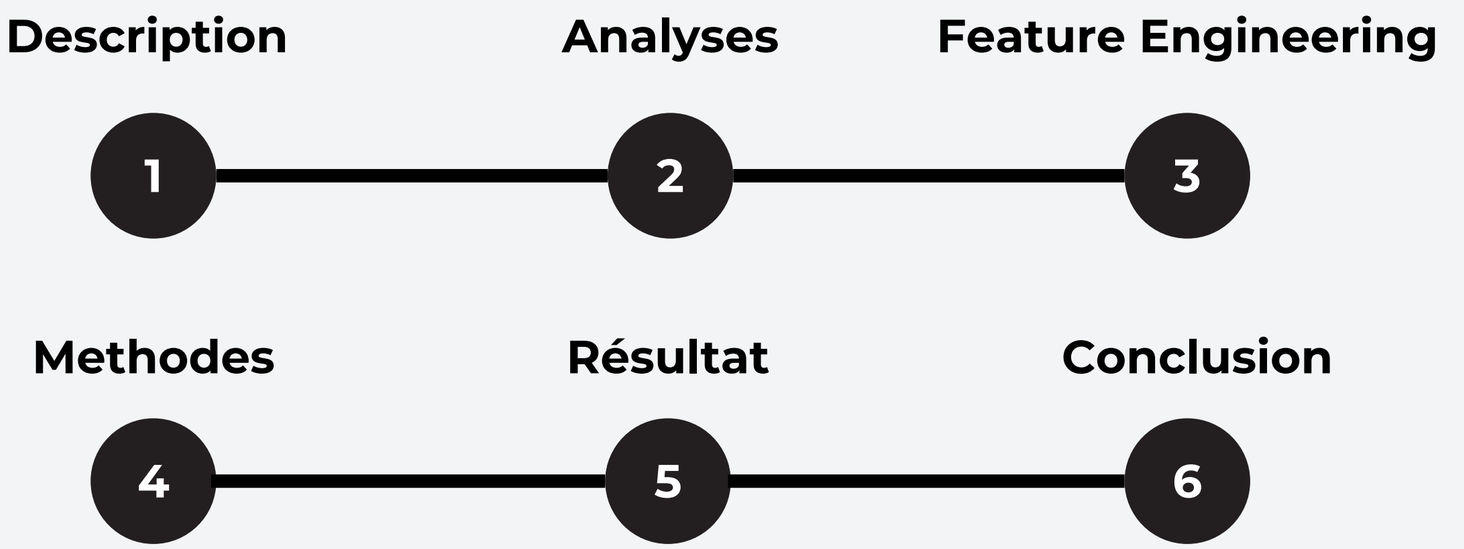
Supervisé par : Fabien Viger

Université Paris-Cité
Master Informatique Fondamentale - Parcous DATA
Projet - Fouille de données et aide à la décision
Année : 2025/2026

Code

Université Paris Cité

SOMMAIRE


Description
Analyses
Feature Engineering
1
2
3
Methodes
Résultat
Conclusion
4
5
6

Description

- Il s’agit d’une compétition Kaggle faisant partie de la série Kaggle Playground (S5 E11).
- **Objective :** la prédiction de la probabilité qu’un emprunteur rembourse son prêt (loan_paid_back). Il s’agit donc d’un probleme d’une **classification binaire. (1/0)**
- **Dataset :**
  - **Démographie de l’Emprunteur :** Gender, Marital Status, Education Level
  - **Informations Financières :** Annual Income, Employement Status, DBI Ratio, Credit Score.
    - Credit Score (300–579 : Poor, 580–669 : Fair, 670–739 : Good, 740+ : Excellent)
  - **Informations sur le Prêt :** Loan Amount, Loan Purpose, Interest Rate, Grade Subgrade (A1, B2, C1, F5, …)
- **Métrique :** AUC-ROC (Area under the curve)

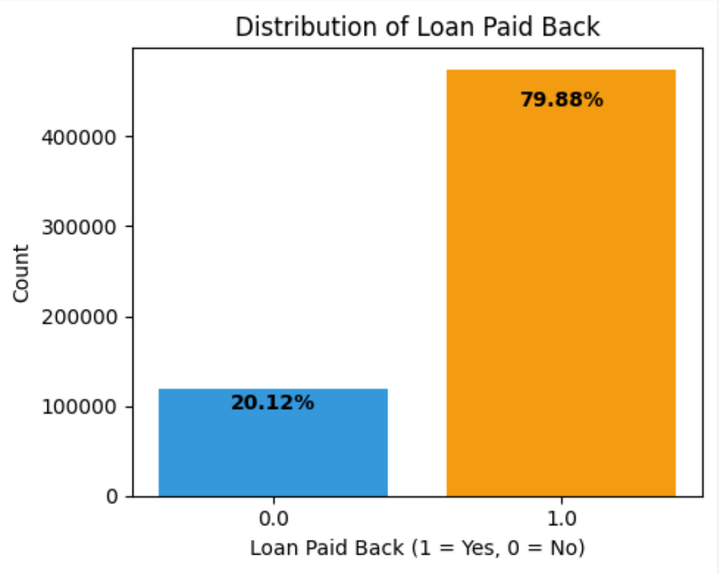
Analyse Exploratoire des Données (EDA)

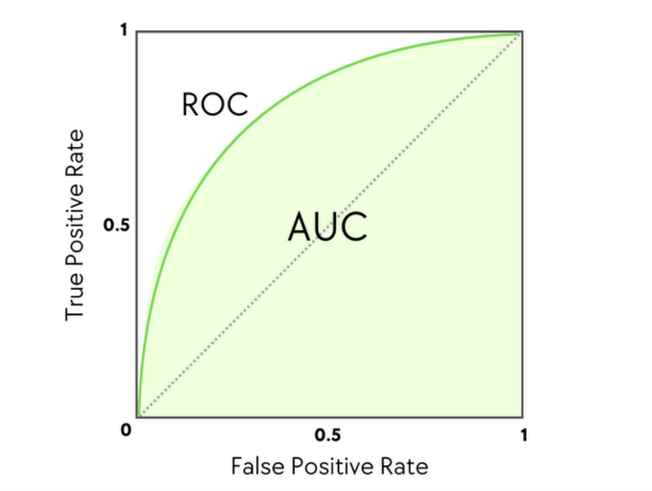
Analyses sur la variable cible

- La variable cible est **déséquilibrée** : ~80 % remboursés (1), 20 % en défaut (0).

- Un **modèle naïf** qui prédit toujours le remboursement atteindrait **~80 % de précision**, mais ne distingue pas le risque.
- Pour ce même modèle naïf, le score ROC-AUC serait de **0,5 (50 %)**, ce qui correspond à une performance équivalente au hasard.

- Le **ROC-AUC** est choisi comme métrique principale, car il mesure mieux la capacité à **différencier** les prêts remboursés et en défaut.

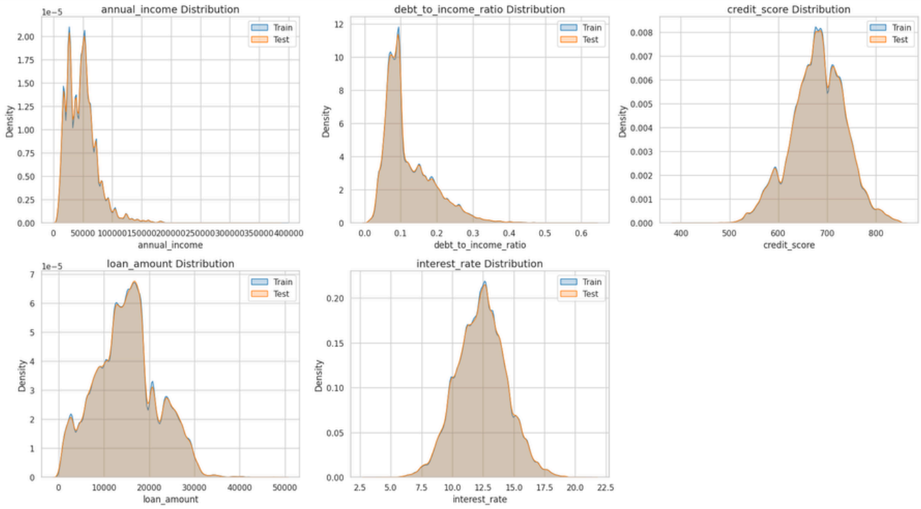




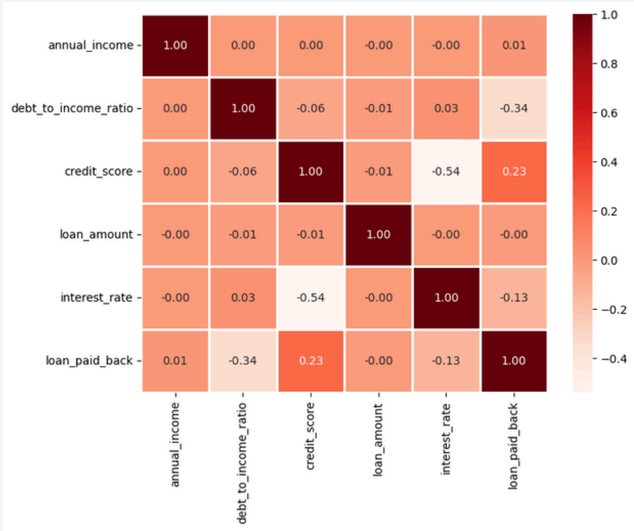
Analyse Exploratoire des Données (EDA)

Analyses sur les variables numériques

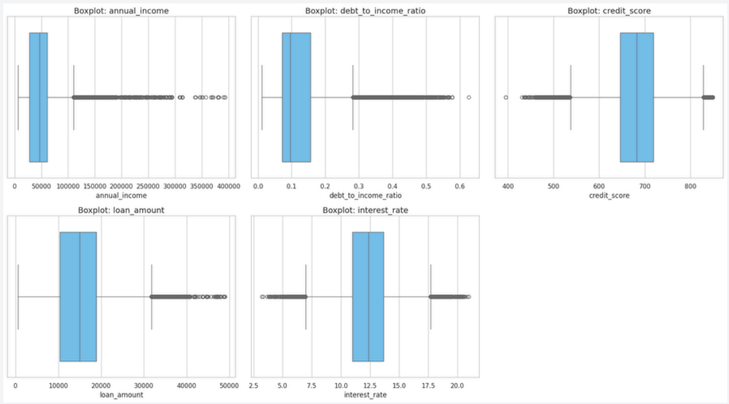
Distributions Drift (Train vs Test)



Correlation Matrix



Boxplots



Outliers ratio

	Feature	Outlier Count	Outlier Percentage (%)
1	debt_to_income_ratio	17556	2.96%
0	annual_income	15917	2.68%
2	credit_score	5901	0.99%
4	interest_rate	5136	0.86%
3	loan_amount	2902	0.49%

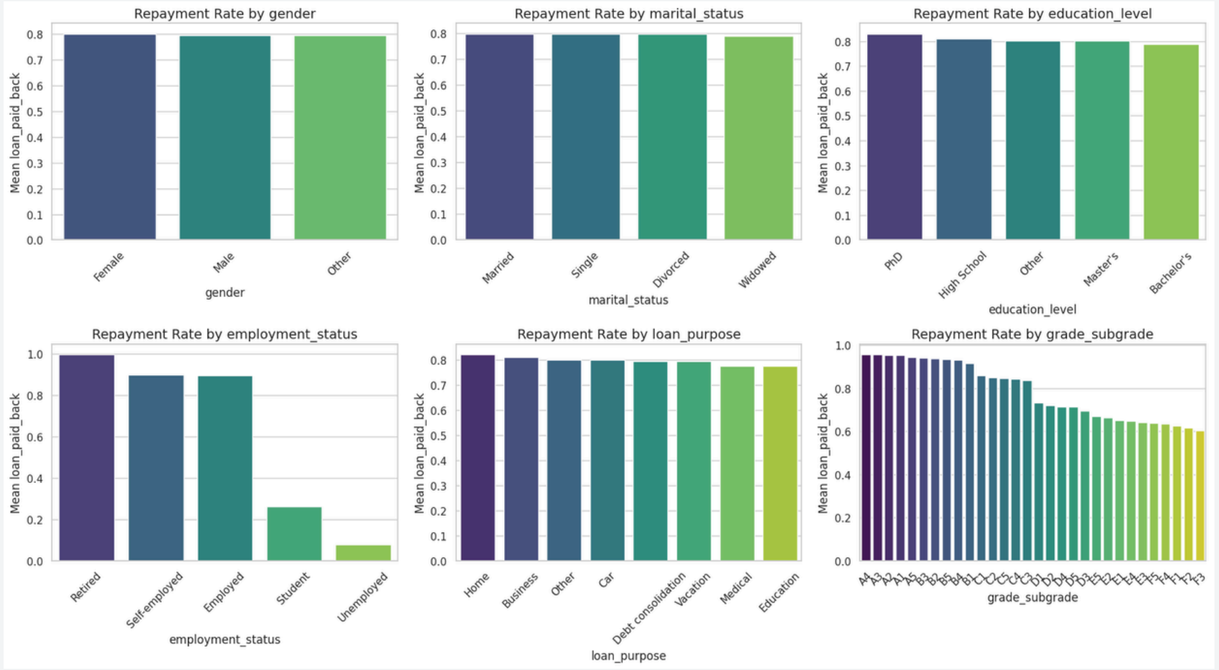
Analyse Exploratoire des Données (EDA)

Analyses sur les variables catégorielles

- Teste statistique **CHI-2** pour voir l'association des variables categorielles avec la variable cible

	Feature	Chi2 Statistic	Degrees of Freedom	p-value
3	employment_status	256259.86	4	0.0000e+00
5	grade_subgrade	30871.16	29	0.0000e+00
2	education_level	383.43	4	1.0552e-81
4	loan_purpose	391.33	7	1.7259e-80
0	gender	32.81	2	7.4954e-08
1	marital_status	4.12	3	2.4856e-01

- Taux de remboursement par variable catégorielle



Features Engineering

- Supprimer les outliers
- Supprimer les colonnes inutiles comme “Marital status”
- Normaliser les valeurs
  - pour les variables catégorielles : OneHot Encoding
  - pour les variables numeriques : StandardScaler

```python
def remove_outliers_iqr(df, numerical_cols, factor=1.5):
    df_clean = df.copy()

    for col in numerical_cols:
        Q1 = df_clean[col].quantile(0.25)
        Q3 = df_clean[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - factor * IQR
        upper_bound = Q3 + factor * IQR

        df_clean = df_clean[
            (df_clean[col] >= lower_bound) &
            (df_clean[col] <= upper_bound)
        ]

    return df_clean

train_df = remove_outliers_iqr(train, num_cols)
test_df = remove_outliers_iqr(test, num_cols)
```

```python
scaler = StandardScaler()
train_df[num_cols] = scaler.fit_transform(train_df[num_cols])
test_df[num_cols] = scaler.fit_transform(test_df[num_cols])
```

```python
import pandas as pd

train_df = pd.get_dummies(
    train_df,
    columns=cat_cols,
    drop_first=True
)

bool_cols = train_df.select_dtypes(include='bool').columns
train_df[bool_cols] = train_df[bool_cols].astype(int)
train_df.head()
```

- Ajouter plus des variables d’interactions :

```python
def create_advanced_features(df):
    # Core affordability
    df['income_loan_ratio'] = df['annual_income'] / (df['loan_amount'] + 1)
    df['loan_to_income'] = df['loan_amount'] / (df['annual_income'] + 1)

    # Debt metrics
    df['total_debt'] = df['debt_to_income_ratio'] * df['annual_income']
    df['available_income'] = df['annual_income'] * (1 - df['debt_to_income_ratio'])
    df['debt_burden'] = df['debt_to_income_ratio'] * df['loan_amount']

    # Payment analysis
    df['monthly_payment'] = df['loan_amount'] * df['interest_rate'] / 1200
    df['payment_to_income'] = df['monthly_payment'] / (df['annual_income'] / 12 + 1)
    df['affordability'] = df['available_income'] / (df['loan_amount'] + 1)

    #Custom Risk scoring
    df['default_risk'] = (df['debt_to_income_ratio'] * 0.40 +
                          (850 - df['credit_score']) / 850 * 0.35 +
                          df['interest_rate'] / 100 * 0.25)
```

Methodes (Modeling)

- Tester plusieurs méthodes, allant d’algorithmes simples à des algorithmes plus avancés.
- Ajuster différents paramètres pour chaque algorithme afin de sélectionner la meilleure configuration.
- Appliquer une validation croisée (CV) pour chaque modèle afin de garantir sa capacité de généralisation.

```python
# Définir la grille d'hyperparamètres
param_grid = {
    'C': [0.01, 0.1, 1],
    'penalty': ['l1', 'l2', 'elasticnet'],
    'class_weight': ['balanced']
}

# Stratified K-Fold pour GridSearch
cv_grid = StratifiedKFold(n_splits=8, shuffle=True, random_state=42)

# GridSearchCV avec scoring ROC-AUC
grid = GridSearchCV(model, param_grid, cv=cv_grid, scoring='roc_auc', n_jobs=-1)
grid.fit(X, y)

# Meilleurs paramètres et score
print("Best params:", grid.best_params_)
print("Best ROC-AUC (CV mean):", round(grid.best_score_, 4))

# Sauvegarder le meilleur modèle
joblib.dump(grid.best_estimator_, "models/best_logreg_model.joblib")
print("Modèle sauvegardé sous 'models/best_logreg_model.joblib'")
```

Methodes simples

Regression Logistique

```python
# Définir le modèle
model = LogisticRegression(max_iter=1000, solver='saga')

# Définir la grille d'hyperparamètres
param_grid = {
    'C': [0.01, 0.1, 1],
    'penalty': ['l1', 'l2', 'elasticnet'],
    'class_weight': ['balanced']
}
```

- penalty (L1 : lasso, L2: ridge, Elasticnet : hyprid).
- parametre de regularisation C.

Arbre de décision

```python
# Définir le modèle de base
model = DecisionTreeClassifier(random_state=42, class_weight='balanced')

# Grille d'hyperparamètres à tester
param_grid = {
    'max_depth': [4, 6, 8,],
    'min_samples_split': [10, 50],
    'min_samples_leaf': [10, 50],
}
```

- max_depth, min_sample_split, min_sample_leaf

Naive Bayes

```python
# Définir le modèle
model = BernoulliNB()

# Définir le CV
cv = StratifiedKFold(n_splits=8, shuffle=True, random_state=42)

# Cross-validation sur ROC-AUC
scores = cross_val_score(model, X, y, cv=cv, scoring='roc_auc')

print("ROC-AUC scores per fold:", np.round(scores, 4))
print("Mean ROC-AUC:", np.round(scores.mean(), 4))
```

- Bernoulli vs Gaussian

Methodes (Modeling)

Méthodes Avancés

Bagging (Random Forrest)

- Parametres à varier : estimators, max_depth, min_samples_split

```
# Définir le modèle de base
model = RandomForestClassifier(random_state=42, class_weight='balanced', n_jobs=-1)

# Grille d'hyperparamètres à tester
param_grid = {
    'n_estimators': [500, 1000],
    'max_depth': [6, 8],
    'min_samples_split': [10, 50],
    'min_samples_leaf': [50, 100]
}
```

Boosting

- Algorithmes : XGBoost, LightGBM, CatBoost
- Parametres a varier : learning rate, estimators

```
# Définir le modèle de base
model = LGBMClassifier(
    random_state=42,
    n_jobs=-1,
    max_depth=6,
    eval_metric='auc',
    use_label_encoder=False,
    verbose=-1
)

# Grille d'hyperparamètres à tester
param_grid = {
    'n_estimators': [500, 1000],
    'learning_rate': [0.05, 0.1]
}
```

Stacking

- Combiner plusieurs modèles performants afin d’exploiter leurs forces complémentaires.
  - les trois meilleurs modèles identifiés jusqu’à présent sont utilisés comme **modèles de niveau 1 (base learners)**.
  - leurs prédictions servent ensuite d’entrées à **un modèle de niveau 2 (meta-learner)**

```
# Définir les modèles de base
estimators = [
    ('lgbm', LGBMClassifier(
        n_estimators=1000,        # nombre d’arbres
        max_depth=6,              # profondeur maximale
        learning_rate=0.1,        # taux d'apprentissage
        eval_metric='auc',        # métrique pour XGBoost
        use_label_encoder=False, # éviter warning
        random_state=42,
        n_jobs=-1
    )),
    ('xgb', XGBClassifier(
        n_estimators=1000,        # nombre d’arbres
        max_depth=8,              # profondeur maximale
        learning_rate=0.05,       # taux d'apprentissage
        eval_metric='auc',        # métrique pour XGBoost
        use_label_encoder=False, # éviter warning
        random_state=42,
        n_jobs=-1
    )),
    ('cat', CatBoostClassifier(
        n_estimators=1000,        # nombre d’arbres
        max_depth=8,              # profondeur maximale
        learning_rate=0.1,        # taux d'apprentissage
        eval_metric='AUC',        # métrique pour CatBoost
        random_state=42
    ))
]

# Méta-modèle
meta_model = LogisticRegression(max_iter=1000, penalty='l2', solver='lbfgs', class_weight='balanced')

# Stacking Classifier
stack_model = StackingClassifier(
    estimators=estimators,
    final_estimator=meta_model,
    cv=5,                # pour créer les features pour le méta-modèle
    n_jobs=-1,
    passthrough=True     # les features originales passent aussi au méta-modèle
)
```

Resultats

Méthodes Simples

```
Best params: {'max_depth': 8, 'min_samples_leaf': 50, 'min_samples_split': 10}
Best ROC-AUC (CV mean): 0.9115
Modèle sauvegardé sous 'best_tree_model.joblib'
```

```
Best params: {'C': 0.1, 'class_weight': 'balanced', 'penalty': 'l1'}
Best ROC-AUC (CV mean): 0.9112
Modèle sauvegardé sous 'models/best_logreg_model.joblib'
```

```
ROC-AUC scores per fold: [0.8819 0.8769 0.878  0.8774 0.8793 0.8737 0.8792 0.8795]
Mean ROC-AUC: 0.8782
Modèle sauvegardé sous 'models/best_naive_bayes_model.joblib'
```

Méthodes Avancés

```
Best params: {'max_depth': 6, 'n_estimators': 1000, 'learning_rate': 0.1}
Best ROC-AUC (CV mean): 0.9214
Modèle sauvegardé sous 'models/best_lgbm_model.joblib'
```

```
Best params: {'max_depth': 6, 'min_samples_leaf': 50, 'min_samples_split': 10, 'n_estimators': 500}
Best ROC-AUC (CV mean): 0.9011
Modèle sauvegardé sous 'models/best_rf_model.joblib'
```

```
Best params: {'max_depth': 8, 'n_estimators': 1000, 'learning_rate': 0.05}
Best ROC-AUC (CV mean): 0.9195
Modèle sauvegardé sous 'models/best_xgboost_model.joblib'
```

```
Stacking Classifier Results:
ROC-AUC scores per fold: [0.9213 0.9196 0.9188 0.9174 0.9197 0.9171 0.9202 0.9185]
Mean ROC-AUC: 0.9191
```

```
Best params: {'max_depth': 8, 'n_estimators': 1000, 'learning_rate': 0.1}
Best ROC-AUC (CV mean): 0.9213
Modèle sauvegardé sous 'models/best_catboost_model.joblib'
```

Conclusion