Mémoire de Projet d'ingénierie

Filière : Data engineer

Data Mining Project

Prévision des crédits logement

Réalisé par

HALMAOUI Hajar

Encadré par :

ELASRI Ikram

Année universitaire :2023/2024

Table des matières

Introduction 3
1. Data Selection and understanding 3
2. Data preparation and visualisation 5
- **Vérification des valeurs manquantes dans le jeu de données des crédits.** 5
- **Remplissage des valeurs manquantes dans le jeu de données des crédits.** 5
- **Répartition des crédits accordés et refusés** 6
- **Répartition du nombre d'enfants à charge** 7
- **Distribution et boîte à moustaches des revenus des demandeurs** 8
- **Distribution et boîte à moustaches des revenus des co-demandeurs** 9
- **Répartition des variables catégorielles par statut de prêt** 9
- **Matrice de corrélation des variables numériques** 10
- **Encodage one-hot des variables catégorielles** 11
- **Fusion des données catégorielles encodées et des données numériques** 12
- **Séparation des variables cibles et des caractéristiques** 13

3. Modélisation : Classification 14
- **Division des données en ensembles d'entraînement et de test** 14
- **Fonction de traçage de la matrice de confusion** 14
- **Évaluation du modèle de régression logistique** 14
- **Évaluation du modèle SVM** 16
- **Évaluation du modèle Random Forest** 18
- **Évaluation du modèle Gradient Boosting** 19
- **Évaluation du modèle de réseau de neurones artificiels (ANN)** 21

Introduction

Le projet vise à développer un système de prédiction de l'approbation ou du rejet de prêts bancaires en se basant sur un ensemble de caractéristiques des demandeurs. Pour ce faire, plusieurs algorithmes d'apprentissage automatique sont utilisés, notamment la Forêt aléatoire (Random Forest), les Machines à gradient boosté (Gradient Boosting Machines), les Réseaux de neurones artificiels (ANN), la Régression logistique (Logistic Regression) et les Machines à vecteurs de support (SVM). Chaque algorithme est évalué en termes de précision et de rappel pour déterminer celui qui offre les meilleures performances dans la prédiction des décisions de prêt.

1. Data Selection and understanding

Les données utilisées dans notre application proviennent de la plateforme Kaggle. Il s'agit d'une table de donnée 'loan_sanction_train' :

– loan_sanction_train: il contient initialement 615 lignes et ses attributs sont :

- **Loan_ID**: Identifiant unique associé à chaque demande de prêt.
- **Gender**: Genre du demandeur (masculin ou féminin).
- **Married**: Indique si le demandeur est marié ou non (Oui/Non).
- **Dependents**: Nombre de personnes à charge financièrement sur le demandeur.
- **Education**: Niveau d'éducation du demandeur (Gradué ou Non Gradué).
- **Self_Employed**: Indique si le demandeur est travailleur indépendant ou non (Oui/Non).
- **ApplicantIncome**: Revenu mensuel de l'emprunteur principal en termes de montant.
- **CoapplicantIncome**: Revenu mensuel du co-emprunteur en termes de montant.
- **LoanAmount**: Montant du prêt demandé en termes de devise.
- **Loan_Amount_Term**: Durée du prêt en mois.
- **Credit_History**: Historique de crédit du demandeur (1 pour bon historique de crédit, 0 pour mauvais historique de crédit).
- **Property_Area**: Zone de propriété où se situe la propriété pour laquelle le prêt est demandé (Rurale, Semi-urbaine, Urbaine).
- **Loan_Status**: Statut de l'approbation du prêt (Oui/Non), où "Oui" indique que le prêt a été approuvé et "Non" indique que le prêt n'a pas été approuvé.

```
df_credit
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

614 rows × 13 columns

```
df_credit
```

ried	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...
No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

Le jeu de données est accessible à travers le lien suivant :
HTTPS://WWW.KAGGLE.COM/CODE/SHIVKUMARMEHMI/LOAN-APPROVAL-PREDICTION/INPUT

2. Data preparation and visualisation

➢ **Vérification des valeurs manquantes dans le jeu de données des crédits.**

```
df_credit.isnull().sum()
```

```
Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

Cette opération montre le nombre de valeurs manquantes dans chaque colonne du jeu de données des crédits. Les colonnes telles que 'Gender', 'Married', 'Dependents', 'Self_Employed', 'LoanAmount', 'Loan_Amount_Term' et 'Credit_History' ont des valeurs manquantes. Ces valeurs manquantes devront être traitées lors de la préparation des données pour l'analyse et la modélisation.

➢ **Remplissage des valeurs manquantes dans le jeu de données des crédits.**

```
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])
df['Married'] = df['Married'].fillna(df['Married'].mode()[0])
df['Dependents'] = df['Dependents'].fillna(df['Dependents'].mode()[0])
df['Self_Employed'] = df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].mode()[0])
```

```
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].median())
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].median())
```

Cette étape consiste à remplir les valeurs manquantes dans les colonnes pertinentes du jeu de données des crédits. Les colonnes 'Gender', 'Married', 'Dependents', 'Self_Employed' et 'Credit_History' sont remplies avec le mode de chaque colonne, tandis que les colonnes 'LoanAmount' et 'Loan_Amount_Term' sont remplies avec leur médiane respective. Cela permet de préparer les données pour l'analyse et la modélisation en assurant que toutes les colonnes ont des valeurs valides.

```
df.isnull().sum()
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

- **Répartition des crédits accordés et refusés**

```
df['Loan_Status'].value_counts()
```

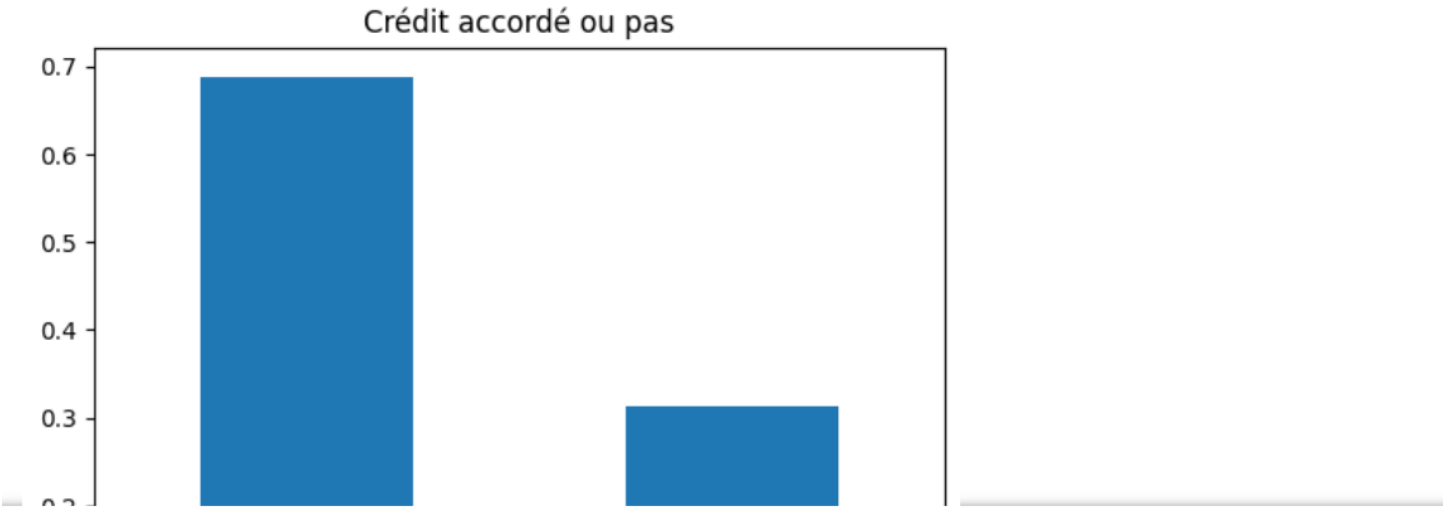
```
Loan_Status
Y    422
N    192
Name: count, dtype: int64
```

```
df['Loan_Status'].value_counts(normalize=True)*100
```

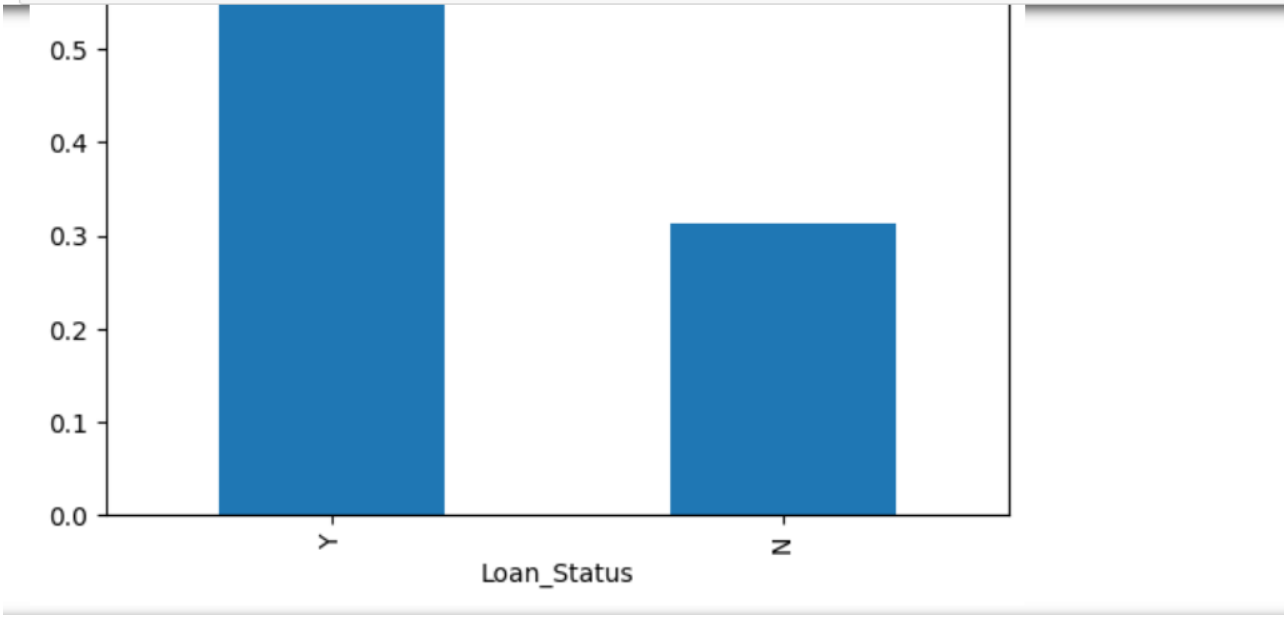
```
Loan_Status
Y    68.729642
N    31.270358
Name: proportion, dtype: float64
```

```
df['Loan_Status'].value_counts(normalize=True).plot.bar(title='Crédit accordé ou pas')
```

```
<Axes: title={'center': 'Crédit accordé ou pas'}, xlabel='Loan_Status'>
```



```
df['Loan_Status'].value_counts(normalize=True).plot.bar(title='Crédit accordé ou pas')
```



Cette visualisation présente la répartition des crédits accordés (Y) et refusés (N) dans le jeu de données. Sur un total de 614 entrées, 422 crédits ont été accordés, représentant environ 68.73% du total, tandis que 192 crédits ont été refusés, représentant environ 31.27% du total. Ce graphique à barres offre une vue claire de la répartition des décisions de crédit dans le jeu de données.

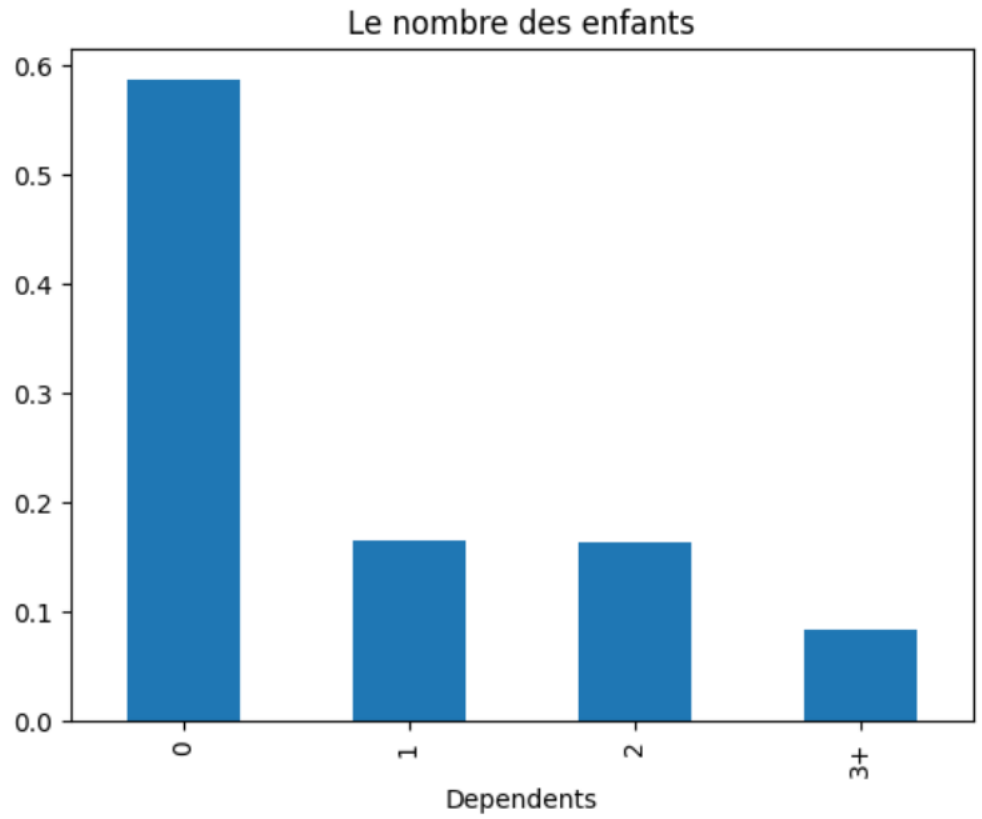
- **Répartition du nombre d'enfants à charge**

```
df['Dependents'].value_counts(normalize=True)*100
```

```
Dependents
0     58.631922
1     16.612378
2     16.449511
3+     8.306189
Name: proportion, dtype: float64
```

```
df['Dependents'].value_counts(normalize=True).plot.bar(title='Le nombre des enfants')
```

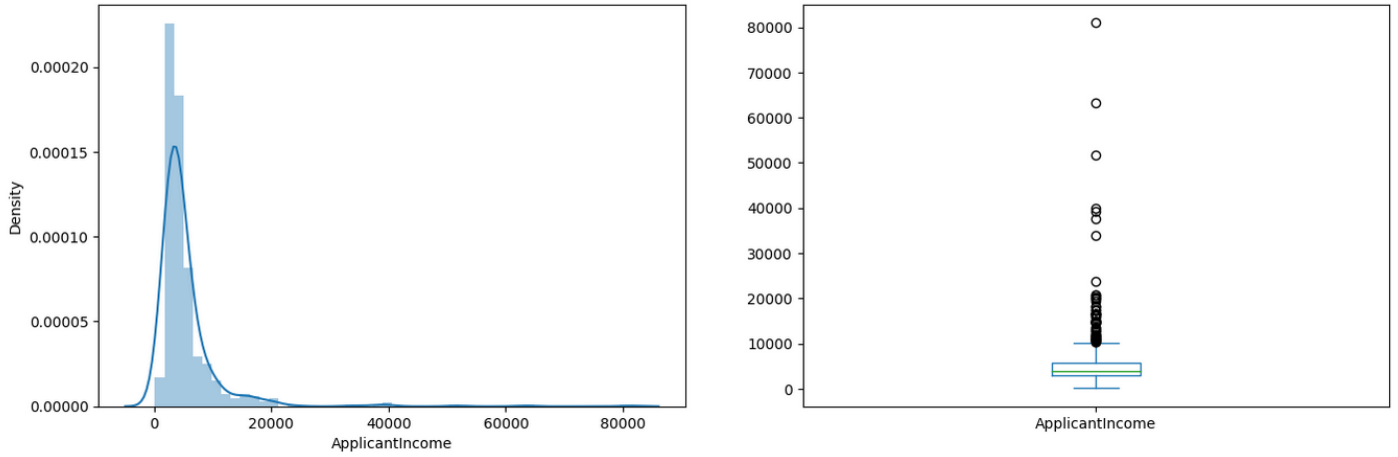
```
<Axes: title={'center': 'Le nombre des enfants'}, xlabel='Dependents'>
```



Ce graphique à barres présente la répartition du nombre d'enfants à charge pour les demandeurs de prêt. Environ 58.63% des demandeurs n'ont aucun enfant à charge, 16.61% ont un enfant à charge, 16.45% en ont deux, et 8.31% ont trois enfants ou plus. Cette visualisation permet de comprendre la composition familiale des demandeurs de prêt dans le jeu de données.

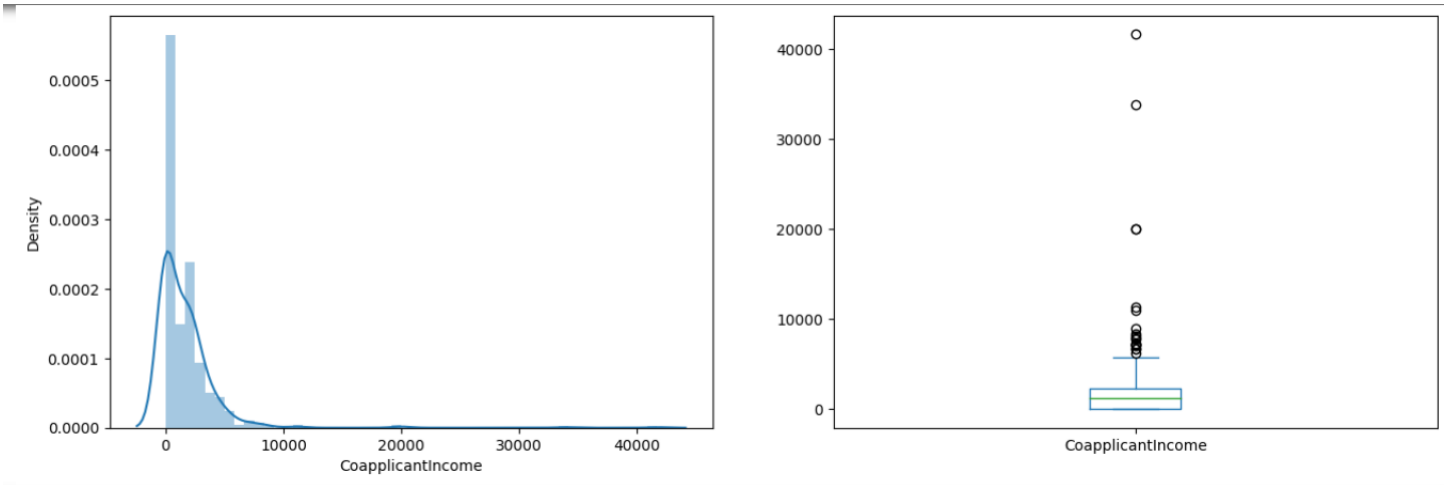
- **Distribution et boîte à moustaches des revenus des demandeurs**

```
plt.figure(1)

plt.subplot(121)
sns.distplot(df['ApplicantIncome'])

plt.subplot(122)
df['ApplicantIncome'].plot.box(figsize=(16,5))

plt.suptitle('')
plt.show()
```



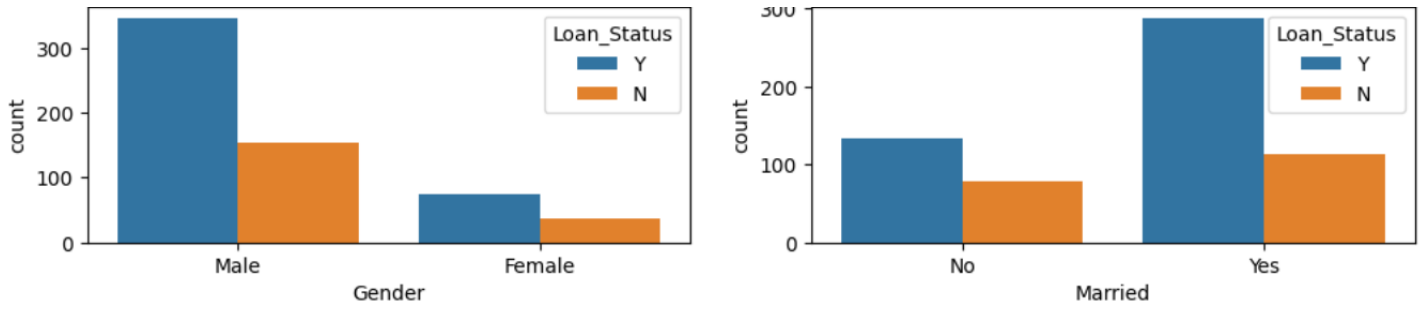
Ce graphique présente la distribution des revenus des demandeurs de prêt sous forme d'histogramme (gauche) et de boîte à moustaches (droite). L'histogramme montre la répartition des revenus des demandeurs, tandis que la boîte à moustaches met en évidence la médiane, les quartiles et les valeurs aberrantes éventuelles. Cette visualisation permet d'analyser la distribution et la dispersion des revenus des demandeurs dans le jeu de données.

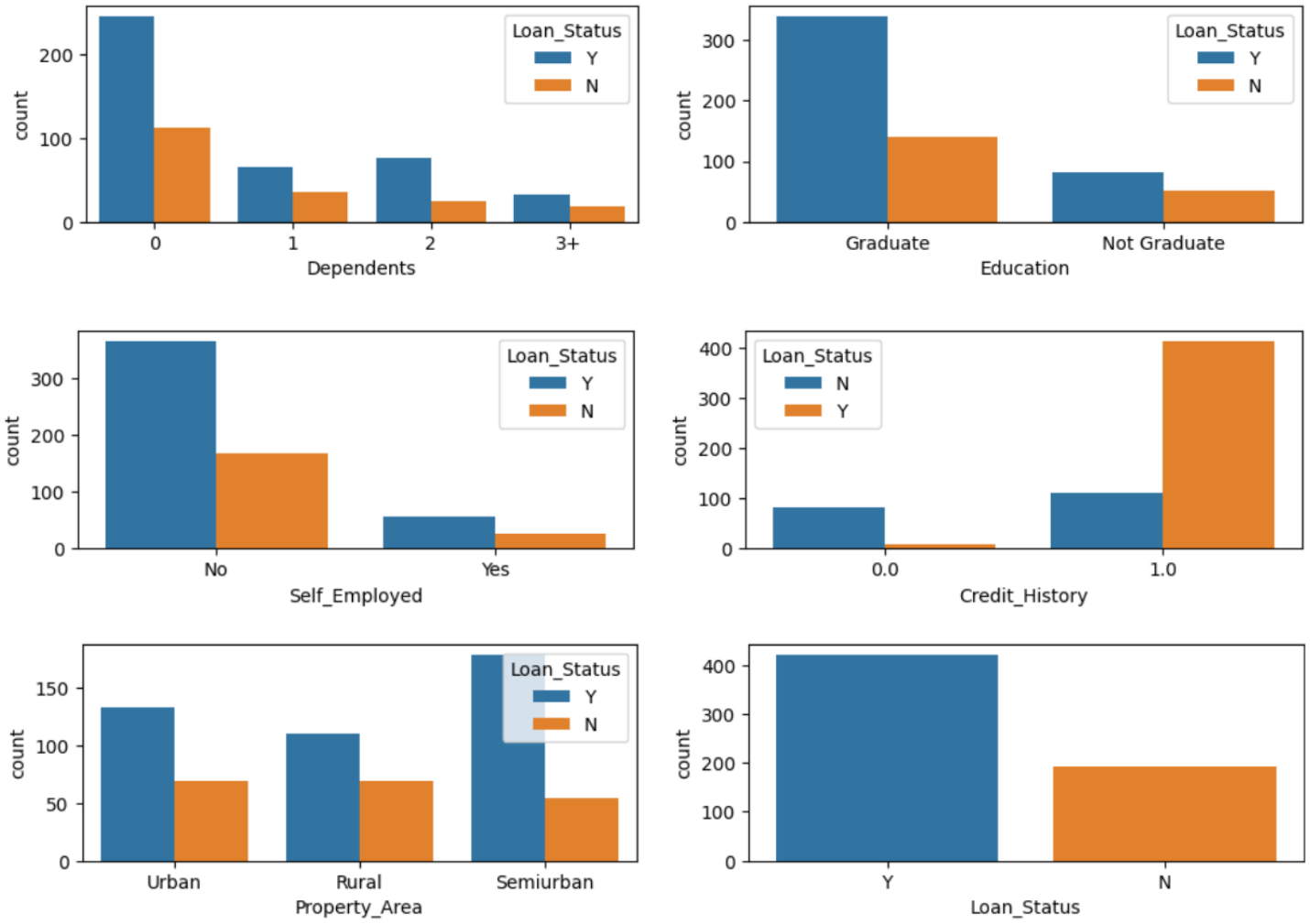
- **Distribution et boîte à moustaches des revenus des co-demandeurs**



Ce graphique présente la distribution des revenus des co-demandeurs de prêt sous forme d'histogramme (gauche) et de boîte à moustaches (droite). L'histogramme montre la répartition des revenus des co-demandeurs, tandis que la boîte à moustaches met en évidence la médiane, les quartiles et les valeurs aberrantes éventuelles. Cette visualisation permet d'analyser la distribution et la dispersion des revenus des co-demandeurs dans le jeu de données.

- **Répartition des variables catégorielles par statut de prêt**

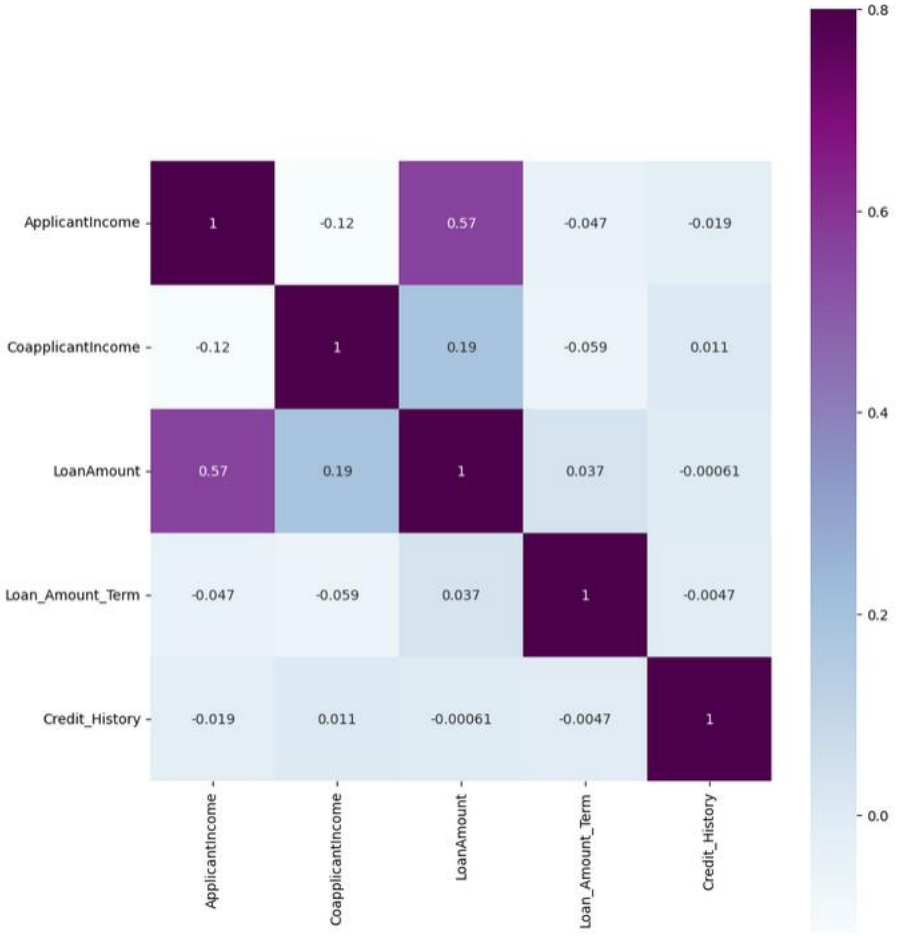




Ces graphiques présentent la répartition des variables catégorielles par statut de prêt. Chaque sous-graphique représente une variable catégorielle et montre le nombre de prêts accordés ou refusés dans chaque catégorie. Les barres colorées indiquent le nombre de prêts accordés (Y) et refusés (N) pour chaque catégorie de variable. Cette visualisation permet de comparer visuellement la probabilité d'obtenir un prêt dans différentes catégories de variables.

- **Matrice de corrélation des variables numériques**

```
#correlation des variables numerique
numeric_columns = df.select_dtypes(include=['int64', 'float64'])
matrix=numeric_columns.corr()
f,ax=plt.subplots(figsize=(10,12))
sns.heatmap(matrix,vmax=.8,square=True,cmap='BuPu',annot=True)
plt.show()
```



La heatmap ci-dessus illustre la corrélation entre les variables numériques du jeu de données. Les valeurs plus proches de 1 indiquent une corrélation positive, tandis que les valeurs plus proches de -1 indiquent une corrélation négative. Les valeurs proches de zéro indiquent une faible corrélation. Cette visualisation permet de détecter les relations linéaires entre les variables numériques, ce qui peut être utile pour comprendre l'impact de chaque variable sur les autres dans le contexte du problème de prêt.

➢ Encodage one-hot des variables catégorielles

```
df_cat=pd.get_dummies(df_cat,drop_first=True)
df_cat
```

	Credit_History	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Pr
0	1.0	True	False	False	False	False	False	False	False	
1	1.0	True	True	True	False	False	False	False	False	
2	1.0	True	True	False	False	False	False	True	False	
3	1.0	True	True	False	False	False	True	False	False	
4	1.0	True	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	
609	1.0	False	False	False	False	False	False	False	False	
610	1.0	True	True	False	False	True	False	False	False	
611	1.0	True	True	True	False	False	False	False	False	
612	1.0	True	True	False	True	False	False	False	False	
613	0.0	False	False	False	False	False	False	True	True	

614 rows × 11 columns

```
df_cat=pd.get_dummies(df_cat,drop_first=True)
df_cat
```

Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Property_Area_Urban	Loan_Status_Y
False	False	False	False	False	False	False	True	True
True	True	False	False	False	False	False	False	False
True	False	False	False	False	True	False	True	True
True	False	False	False	True	False	False	True	True
False	False	False	False	False	False	False	True	True
...	...	...	...	...	...	...	...	...
False	False	False	False	False	False	False	False	True
True	False	False	True	False	False	False	False	True
True	True	False	False	False	False	False	True	True
True	False	True	False	False	False	False	True	True
False	False	False	False	False	True	True	False	False

Le DataFrame df_cat a été transformé en un ensemble de variables indicatrices (dummy variables) à l'aide de l'encodage one-hot, avec la variable originale étant "drop_first".

➢ Fusion des données catégorielles encodées et des données numériques

```
df_encoded=pd.concat([df_cat,df_num],axis=1).astype(int)
df_encoded
```

ents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Property_Area_Urban	Loan_Status_Y	ApplicantIncome	CoapplicantIncome	LoanAmount
0	0	0	0	1	1	5849	0	128
0	0	0	0	0	0	4583	1508	128
0	0	1	0	1	1	3000	0	66
0	1	0	0	1	1	2583	2358	120
0	0	0	0	1	1	6000	0	141
...	...	...	...	...	...	...	...	...
0	0	0	0	0	1	2900	0	71
1	0	0	0	0	1	4106	0	40
0	0	0	0	1	1	8072	240	253
0	0	0	0	1	1	7583	0	187
0	0	1	1	0	0	4583	0	133

➢ Séparation des variables cibles et des caractéristiques

```
y=df_encoded['Loan_Status_Y']
y
```

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status_Y, Length: 614, dtype: int32
```

```
X=df_encoded.drop('Loan_Status_Y',axis=1)
X
```

	Credit_History	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	P
0	1	1	0	0	0	0	0	0	0	
1	1	1	1	1	0	0	0	0	0	
2	1	1	1	0	0	0	0	1	0	
3	1	1	1	0	0	0	1	0	0	
4	1	1	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	

Les variables cibles ont été extraites du DataFrame df_encoded et stockées dans la série y. Ces variables représentent les étiquettes de classe indiquant si un prêt a été accordé ou non (1 pour "oui" et 0 pour "non"). Les caractéristiques ont été extraites en supprimant la colonne Loan_Status_Y du DataFrame df_encoded, ce qui donne le DataFrame X contenant les caractéristiques utilisées pour prédire le statut du prêt.

3. Modélisation : Classification

➢ **Division des données en ensembles d'entraînement et de test**

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=6)
```

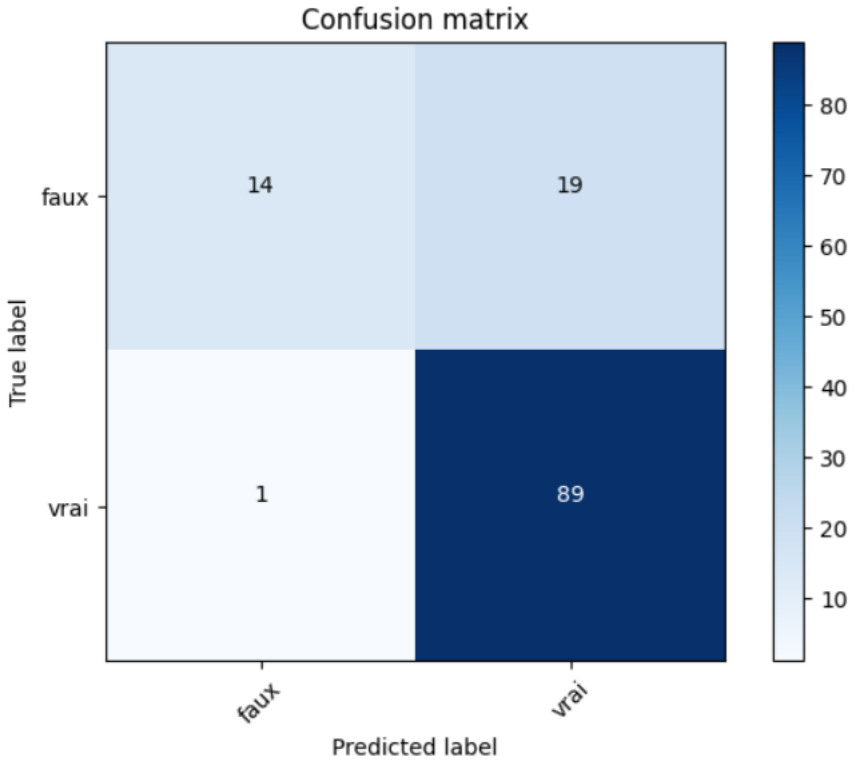
➢ **Fonction de traçage de la matrice de confusion**

```
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",verticalalignment = 'bottom',
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

➢ **Évaluation du modèle de régression logistique**

```python
from sklearn.linear_model import LogisticRegression

# Instantiate the Logistic Regression classifier
clf_lr = LogisticRegression()

# Train the Logistic Regression classifier
clf_lr.fit(X_train, y_train)

# Make predictions on the test set using Logistic Regression
pred_lr = clf_lr.predict(X_test)

# Calculate accuracy on the test set for Logistic Regression
accuracy_lr = accuracy_score(y_test, pred_lr)
print("Accuracy (Logistic Regression):", accuracy_lr)

# Calculate the confusion matrix for Logistic Regression
conf_mat_lr = confusion_matrix(y_test, pred_lr)

# Display the confusion matrix for Logistic Regression
plot_confusion_matrix(conf_mat_lr, classes=['faux', 'vrai'])

plt.savefig('logistic_regression_mat.png')
plt.show()
# Calculate precision and recall for Logistic Regression
precision_lr = precision_score(y_test, pred_lr)
recall_lr = recall_score(y_test, pred_lr)

print("Precision (Logistic Regression):", precision_lr)
print("Recall (Logistic Regression):", recall_lr)
```

```
Accuracy (Logistic Regression): 0.8373983739837398
Confusion matrix, without normalization
```



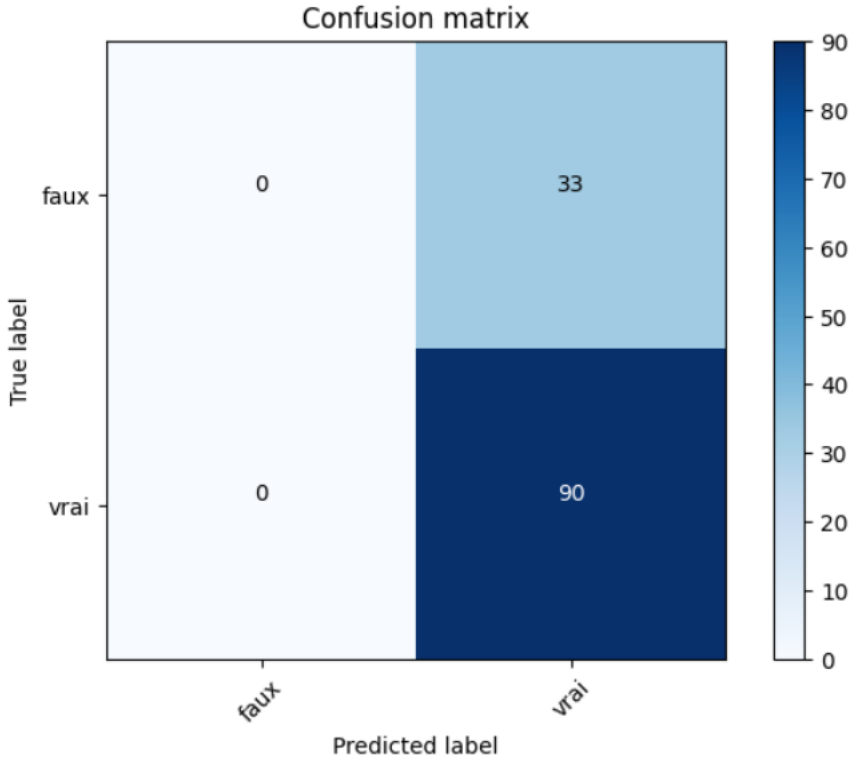

```
Precision (Logistic Regression): 0.8240740740740741
Recall (Logistic Regression): 0.9888888888888889
```

Cette section du code entraîne un modèle de régression logistique sur les données d'apprentissage, puis évalue ses performances en termes d'exactitude, de matrice de confusion, de précision et de rappel sur l'ensemble de test. La précision indique la proportion de vrais positifs parmi les prédictions positives du modèle, tandis que le rappel indique la proportion de vrais positifs identifiés correctement parmi tous les vrais positifs dans les données. Dans ce cas, le modèle de régression logistique a une précision de 0,82 et un rappel de 0,99, ce qui suggère qu'il a bien réussi à identifier les vrais positifs.

- **Évaluation du modèle SVM**

```
from sklearn.svm import SVC

# Instantiate the Support Vector Machine (SVM) classifier
clf_svm = SVC()

# Train the SVM classifier
clf_svm.fit(X_train, y_train)

# Make predictions on the test set using SVM
pred_svm = clf_svm.predict(X_test)

# Calculate accuracy on the test set for SVM
accuracy_svm = accuracy_score(y_test, pred_svm)
print("Accuracy (SVM):", accuracy_svm)

# Calculate the confusion matrix for SVM
conf_mat_svm = confusion_matrix(y_test, pred_svm)

# Display the confusion matrix for SVM
plot_confusion_matrix(conf_mat_svm, classes=['faux', 'vrai'])

plt.savefig('svm_mat.png')
plt.show()
# Calculate precision and recall for SVM
precision_svm = precision_score(y_test, pred_svm)
recall_svm = recall_score(y_test, pred_svm)

print("Precision (SVM):", precision_svm)
print("Recall (SVM):", recall_svm)
```

```
Accuracy (SVM): 0.7317073170731707
Confusion matrix, without normalization
```



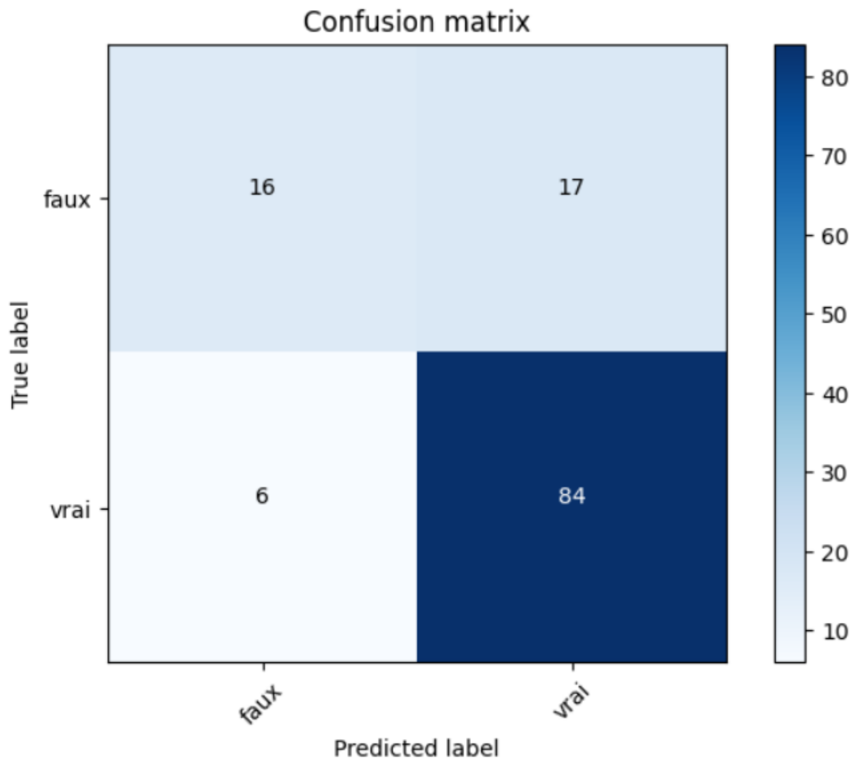

```
Precision (SVM): 0.7317073170731707
Recall (SVM): 1.0
```

Cette partie du code entraîne un modèle de machine à vecteurs de support (SVM) sur les données d'apprentissage, puis évalue ses performances en termes d'exactitude, de matrice de confusion, de précision et de rappel sur l'ensemble de test. La précision et le rappel pour le modèle SVM sont respectivement de 0,73 et 1, ce qui signifie que le modèle a une bonne capacité à identifier les vrais positifs, mais a une précision légèrement inférieure par rapport au modèle de régression logistique précédent.

➢ Évaluation du modèle Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Instantiate the Random Forest classifier
clf_rf = RandomForestClassifier()

# Train the Random Forest classifier
clf_rf.fit(X_train, y_train)

# Make predictions on the test set using Random Forest
pred_rf = clf_rf.predict(X_test)

# Calculate accuracy on the test set for Random Forest
accuracy_rf = accuracy_score(y_test, pred_rf)
print("Accuracy (Random Forest):", accuracy_rf)

# Calculate the confusion matrix for Random Forest
conf_mat_rf = confusion_matrix(y_test, pred_rf)

# Display the confusion matrix for Random Forest
plot_confusion_matrix(conf_mat_rf, classes=['faux', 'vrai'])

plt.savefig('random_forest_mat.png')
plt.show()
precision_rf = precision_score(y_test, pred_rf)
recall_rf = recall_score(y_test, pred_rf)

print("Precision (Random Forest):", precision_rf)
print("Recall (Random Forest):", recall_rf)
```

```
Accuracy (Random Forest): 0.8130081300813008
Confusion matrix, without normalization
```



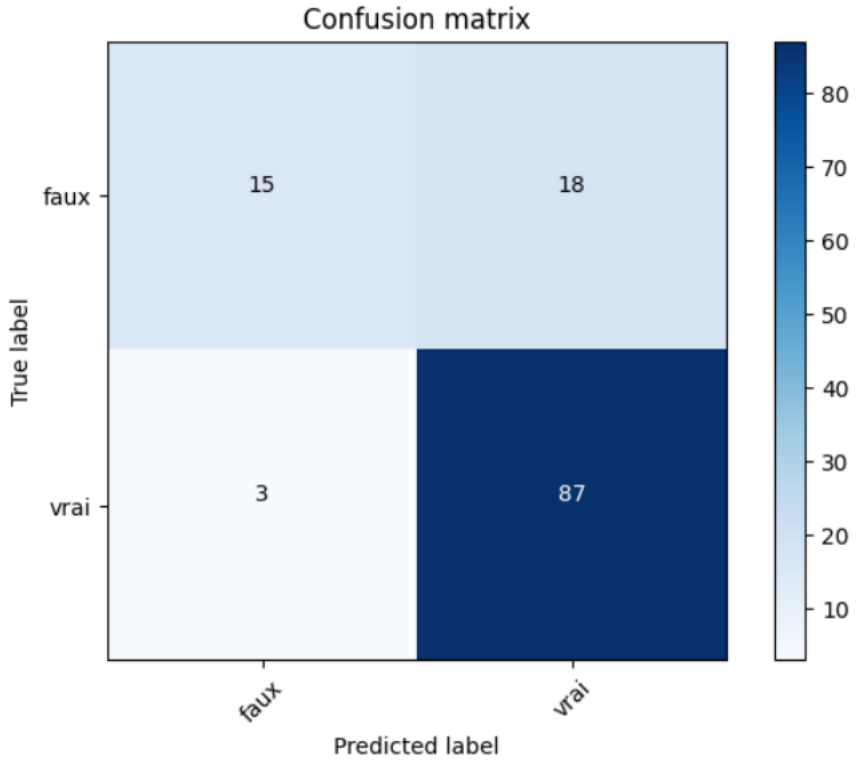

```
Precision (Random Forest): 0.8316831683168316
Recall (Random Forest): 0.9333333333333333
```

Cette partie du code entraîne un modèle de forêt aléatoire sur les données d'apprentissage, puis évalue ses performances en termes d'exactitude, de matrice de confusion, de précision et de rappel sur l'ensemble de test. Le modèle de forêt aléatoire atteint une précision de 0,81 et un rappel de 0,93, montrant de bonnes capacités à prédire à la fois les vrais positifs et les vrais négatifs.

- **Évaluation du modèle Gradient Boosting**

```
from sklearn.ensemble import GradientBoostingClassifier

# Instantiate the Gradient Boosting classifier
clf_gbm = GradientBoostingClassifier()

# Train the Gradient Boosting classifier
clf_gbm.fit(X_train, y_train)

# Make predictions on the test set using GBM
pred_gbm = clf_gbm.predict(X_test)

# Calculate accuracy on the test set for GBM
accuracy_gbm = accuracy_score(y_test, pred_gbm)
print("Accuracy (GBM):", accuracy_gbm)

# Calculate the confusion matrix for GBM
conf_mat_gbm = confusion_matrix(y_test, pred_gbm)

# Display the confusion matrix for GBM
plot_confusion_matrix(conf_mat_gbm, classes=['faux', 'vrai'])

plt.savefig('gbm_mat.png')
plt.show()
# Calculate precision and recall for GBM
precision_gbm = precision_score(y_test, pred_gbm)
recall_gbm = recall_score(y_test, pred_gbm)

print("Precision (GBM):", precision_gbm)
print("Recall (GBM):", recall_gbm)
```

```
Accuracy (GBM): 0.8292682926829268
Confusion matrix, without normalization
```



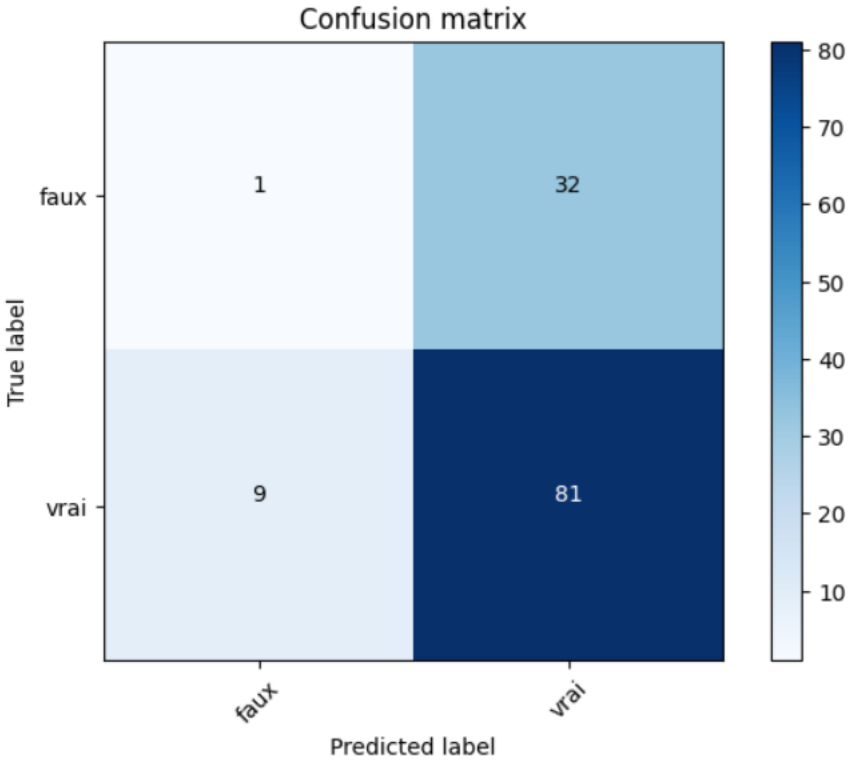

```
Precision (GBM): 0.8285714285714286
Recall (GBM): 0.9666666666666667
```

Cette section du code entraîne un modèle de Gradient Boosting sur les données d'apprentissage, puis évalue ses performances en termes d'exactitude, de matrice de confusion, de précision et de rappel sur l'ensemble de test. Le modèle de Gradient Boosting atteint une précision de 0,83 et un rappel de 0,97, ce qui montre qu'il est capable de bien prédire à la fois les vrais positifs et les vrais négatifs.

- **Évaluation du modèle de réseau de neurones artificiels (ANN)**

```
from sklearn.neural_network import MLPClassifier

# Instantiate the Artificial Neural Network (ANN) classifier
clf_ann = MLPClassifier()

# Train the ANN classifier
clf_ann.fit(X_train, y_train)

# Make predictions on the test set using ANN
pred_ann = clf_ann.predict(X_test)

# Calculate accuracy on the test set for ANN
accuracy_ann = accuracy_score(y_test, pred_ann)
print("Accuracy (ANN):", accuracy_ann)

# Calculate the confusion matrix for ANN
conf_mat_ann = confusion_matrix(y_test, pred_ann)

# Display the confusion matrix for ANN
plot_confusion_matrix(conf_mat_ann, classes=['faux', 'vrai'])

plt.savefig('ann_mat.png')
plt.show()
# Calculate precision and recall for ANN
precision_ann = precision_score(y_test, pred_ann)
recall_ann = recall_score(y_test, pred_ann)

print("Precision (ANN):", precision_ann)
print("Recall (ANN):", recall_ann)
```

```
Accuracy (ANN): 0.6666666666666666
Confusion matrix, without normalization
```




```
Precision (ANN): 0.7168141592920354
Recall (ANN): 0.9
```

Cette partie du code entraîne un modèle de réseau de neurones artificiels (ANN) sur les données d'apprentissage, puis évalue ses performances en termes d'exactitude, de matrice de confusion, de précision et de rappel sur l'ensemble de test. Le modèle ANN atteint une exactitude de 0,67, une précision de 0,72 et un rappel de 0,90. Bien que l'exactitude soit relativement faible par rapport aux autres modèles, le modèle ANN présente une précision et un rappel élevés pour la classe positive, indiquant qu'il peut bien identifier les vrais positifs.

	Logistic Regression	Support Vector Machine (SVM)	Random Forest	Gradient Boosting Machine (GBM)	Artificial Neural Network (ANN)
Accuracy	0,837	0,732	0,813	0,829	0,667
Precision	0,824	0,732	0,832	0,829	0,717
Recall	0,989	1,0	0,933	0,967	0,9

Sur la base de ces résultats, nous pouvons observer que le modèle de régression logistique est celui qui performe le mieux en termes d'exactitude, de précision et de rappel. Il atteint la plus haute exactitude de 0,837 tout en maintenant un bon équilibre entre précision et rappel. Le modèle SVM montre également de bonnes performances avec une exactitude de 0,732, bien qu'il atteigne un rappel parfait au détriment d'une précision légèrement plus faible. Les modèles RandomForest et GBM performent également bien avec des exactitudes de 0,813 et 0,829 respectivement. Cependant, le modèle ANN présente des performances inférieures par rapport aux autres modèles, avec une exactitude de 0,667. Par conséquent, pour cette tâche particulière, le modèle de régression logistique pourrait être le choix le plus adapté.